CS-Lab1-

Ahmed Alhasan, Yash Pawal, Mohsen Pirmoradiyan

1/26/2020

Question 1: Be careful when comparing

```
## [1] "Substraction_is_wrong"
```

```
## [1] "Substraction_is_Correct"
```

- In the first case x1 = 1/3 is not equal to it's mathematical representation, because when it is represented in decimal fraction it continues to infinity, therefore it have to be rounded to the maximum number of digits allowed. In a 64-bit system, Mantissa equals to 52 bits which when transformed to decimal we only have about 16 signicant digits possible.

```r
options(digits = 22)
x1 <- 1/3; x2 <- 1/4
x1
```

```
## [1] 0.3333333333333333148
```

```r
x2
```

```
## [1] 0.25
```

- In the second case the result is correct because both numbers equal to their mathematical value, only integers within the allowable range and fractions whose denominators are factors of base 10 i.e. 1/2 and 1/5 can have exact represenation of their mathematical value.

```r
x1 <- 1; x2 <- 1/2
x1
```

```
## [1] 1
```

```r
x2
```

```
## [1] 0.5
```

- To solve this problem we can either use all.equal() function with appropriate tolernave level to test near equality.

```
x1 <- 1/3; x2 <- 1/4
if (isTRUE(all.equal(x1 - x2, 1/12, tolerance = 0.0001))) {
  print("Substraction_is_Correct")
}else {
  print("Substraction_is_wrong")
}
```

```
## [1] "Substraction_is_Correct"
```

- Or we can round both numbers before testing for equality.

The round() method is a valid way to test near equality. But also note that in R, there is a function call "all.equal" that does this so that you don't need to round it yourself.

```
x1 <- 1/3; x2 <- 1/4
if (round((x1 - x2), 4) == round((1/12),4)) {
  print("Substraction_is_Correct")
}else {
  print("Substraction_is_wrong")
}
```

```
## [1] "Substraction_is_Correct"
```

Question 2: Derivative

```
f <- function(x) {x}
df <- function(x){
  eps <- 10^-15
  (f(x + eps) - f(x)) / eps
}
```

```
df(1)
```

```
## [1] 1.1102230246251565
```

```
df(100000)
```

```
## [1] 0
```

- The true value of both derivatives is equal to 1
- In the first case because of the round-off error, epsilon is not exactly 10^-15 and every time we do arithmatic calculation involving it the result will not resembel the actual number

```
options(digits = 22, scipen = 999)
print(10^-15,digits = 22)
```

```
## [1] 0.000000000000001000000000000000077
```

```r
print((1+10^-15),digits = 22)
```

```
## [1] 1.0000000000000011
```

```r
print((1+10^-15)-1,digits = 22)
```

```
## [1] 0.00000000000000111022302462515654
```

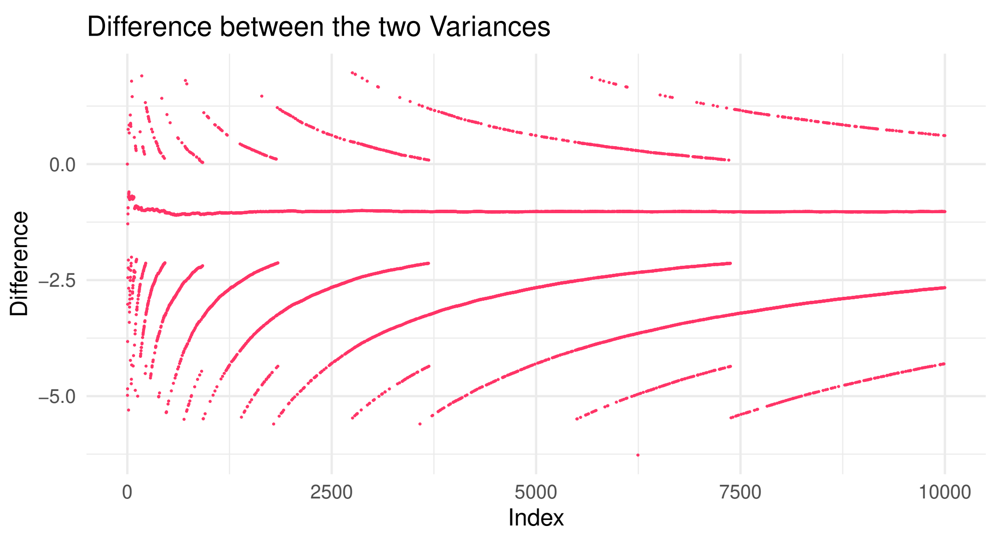
- In the second case because we are adding two digits, one is large and the other very small, the small value of epsilon is neglected because of underflow, therefore the substraction in the nominator becomes 0 and the derivative will yield 0.

Question 3: Variance

```r
myvar <- function(x_vec) {
  n <- length(x_vec)
  (1/(1-n)) * (sum(x_vec^2) - (1/n) * sum(x_vec)^2)
}

x_vec <- rnorm(10000, mean = 10^8, sd = 1)

library(ggplot2)
Y1 <- function(x){
  n <- length(x)
  y <- numeric(length = n)
  for(i in 2:n) {
    X    <- x_vec[1:i]
    y[i] <- myvar(X) - var(X)
  }
  #plot(y, cex = 0.5)
  df <- data.frame(c(1:n),y)
  ggplot(df) +
    geom_point(aes(x = df[,1], y = df[,2]),
               size = 0.01,
               color = "#FF3366") +
    labs(title = "Difference between the two Variances",
         x = "Index",
         y = "Difference")+
    theme_minimal()
}
Y1(x_vec)
```

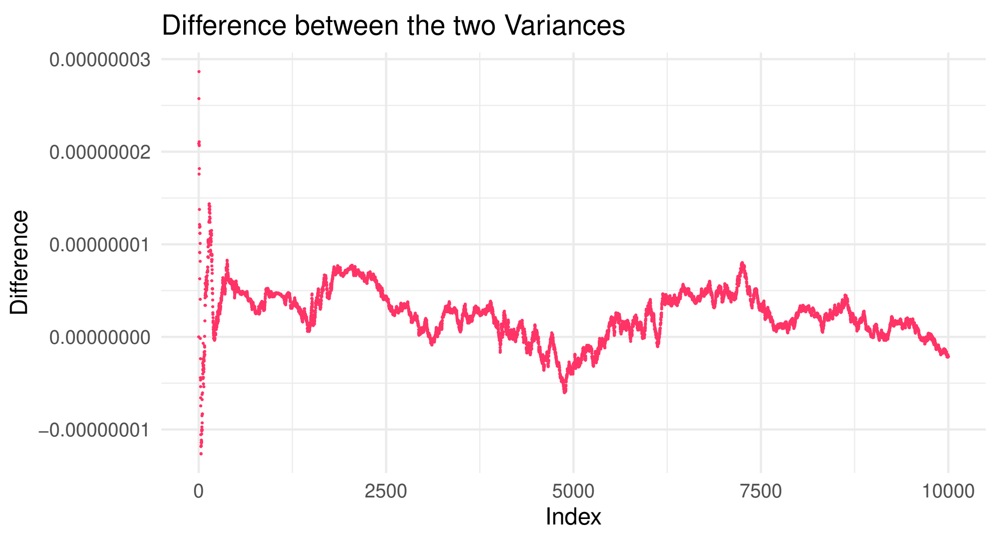


- myvar() function behave wildly because of what is called Catastrophic Cancelation, where the resulted value of sum(x_vec^2) and (1/n) * sum(x_vec)^2 can not be accurately represented by the machine because of the limited storage, so in this case they are rounded to the same number and canceled each other despite them being actually different.

```r
var_YC <- function(v_x){
  ## v_x is a numerical vector of length greater than 2
  ## this function calculates the sample variance
  ## using the Youngs and Cramer algorithm
  T   <- v_x[1]
  RSS <- 0
  n   <- length(v_x)
  for (j in 2:n){
    T   <- T + v_x[j]
    RSS <- RSS + ((j * v_x[j] - T)^2) / (j*(j-1))
  }
  RSS /(n-1)
}

Y2 <- function(x){
  n <- length(x)
  y <- numeric(length = n)
  for(i in 2:n) {
    X    <- x_vec[1:i]
    y[i] <- var_YC(X) - var(X)
  }

  #plot(y, cex = 0.5)
  df <- data.frame(c(1:n),y)
  options(digits = 3)
  ggplot(df) +
    geom_point(aes(x = df[,1], y = df[,2]),
               size = 0.01,
               color = "#FF3366") +
    labs(title = "Difference between the two Variances",
         x = "Index",
         y = "Difference")+
    theme_minimal()
}
Y2(x_vec)
```



- In the Young-Cramer method, it avoid substracting two numbers of almost the same magnitude, so we get a close approximation to the variance function.

Question 4: Linear Algebra

```
options(digits = 5)
tecator <- readxl::read_xls("tecator.xls")

X <- as.matrix(tecator[,-c(1,103)])
y <- as.matrix(tecator[,103])

A <- t(X) %*% X
b <- t(X) %*% y

solve.default(A,b)
```

```
## Error in solve.default(A, b): system is computationally singular: reciprocal condition number = 7.13
```

- The linear system does not have an answer as the matrix A is singular. This Matrix is not invertible. It can happen because of dependency between some variables,i.e, two or more variables are highly correlated. Ths will end in singularity in which the inverse of the matrix does not exist.

```
kappa(A)
```

The intuition and argument is good. But there is one pitfall in your argument: you say that when kapp increases, the upper bound of the relative error in x increases. However, mathematically speaking, that the upper bound of a quantity increases does not imply that the quantity itself increases. This is why in the definition of condition number (for example see Wikipedia) there is a `limsup`.

```
## [1] 1157834236871692
```

- The condition number is very high. If a matrix is singular then its condition number is very large.

For a well-behaved system $Ax = b$, a small change in b $(b + \delta b)$ will cause a relatively small change in $x(x + \delta x)$. It means that if δb is small we expect that the resulting solution $(\tilde{x})$ should be close to x. Such a system is well-conditioned, that is, if $||\delta b||/||b||$ is small, then $||\delta x||/||x||$ is likewise small. By definition:

$$||\delta x||/||x|| \leq ||A||||A^{-1}||||\delta b||/||b||$$

condition number with respect to inversion is $||A||||A^{-1}||$. As the condition number tends to infinity the upper bound of relative change in the solution caused by perturbation $||\delta b||/||b||$ increases. In other words the system is very sensitive to small changes and thus is very susceptible to roundoff error. We do not want this upper bound to be large, so a large condition number is bad.

In this question the condition number is very high and we may conclude that it is an ill-conditioned matrix.

Not trying to be a language police, but these sentences are tautological, verbal redundancy.

```
X_scaled <- scale(X)
y_scaled <- scale(y)

A_new <- t(X_scaled) %*% X_scaled
b_new <- t(X_scaled) %*% y_scaled


kappa(A_new)
```

```
## [1] 490471520662
```

- When we scale the data the round-off error becomes less significant, even though the new condition number is lower it is not necessarily well-conditined, but we have lesser perturbation to deal with.

Discussion: in regard to question 2, there is a discussion within group members whether cancelation happened or not? One of the member believe in "canceletion" and so to avoid this error happening he suggested to sort the numbers ascending: $100000 - 100000 + 1e - 15$ **so that the smaller number at the end of the terms will not be lost, however, this is not agreed by the other members. One of the other member belives that this happens just because of underflow and cancellation is not the case here and rearrangement should not be implemented since the function definition would be changed.**

There is a cancellation and no underflow. Underflow means that the some number is of smaller magnitude that the computer can possibly store. In this case, R is capable of storing 1e-15.

A quick proof is that you can try to make R to compute 1e-15 - 1.4e-15. It will gives you the right answer. If there is an underflow (that is, if R can't store the numbers at all), then the answer won't be correct.

Appendix

```
#Question 1
x1 <- 1/3; x2 <- 1/4
if (x1 - x2 == 1/12) {
  print("Substraction_is_Correct")
}else {
  print("Substraction_is_wrong")
}

x1 <- 1; x2 <- 1/2
if (x1 - x2 == 1/2) {
  print("Substraction_is_Correct")
}else {
  print("Substraction_is_wrong")
}

options(digits = 22)
x1 <- 1/3; x2 <- 1/4
x1
x2

x1 <- 1; x2 <- 1/2
x1
x2

x1 <- 1/3; x2 <- 1/4
if (isTRUE(all.equal(x1 - x2, 1/12, tolerance = 0.0001))) {
  print("Substraction_is_Correct")
}else {
  print("Substraction_is_wrong")
}

x1 <- 1/3; x2 <- 1/4
if (round((x1 - x2), 4) == round((1/12),4)) {
  print("Substraction_is_Correct")
}else {
  print("Substraction_is_wrong")
}
```

```r
## Question 2: Derivative
f <- function(x) {x}
df <- function(x){
  eps <- 10^-15
  (f(x + eps) - f(x)) / eps
}

df(1)
df(10000)

print(10^-15,digits = 22)
print((1+10^-15),digits = 22)
print((1+10^-15)-1,digits = 22)


## Question 3: Variance
myvar <- function(x_vec) {
  n <- length(x_vec)
  (1/(1-n)) * (sum(x_vec^2) - (1/n) * sum(x_vec)^2)
}

x_vec <- rnorm(10000, mean = 10^8, sd = 1)

library(ggplot2)
Y1 <- function(x){
  n <- length(x)
  y <- numeric(length = n)
  for(i in 2:n) {
    X    <- x_vec[1:i]
    y[i] <- myvar(X) - var(X)
  }
  #plot(y, cex = 0.5)
  df <- data.frame(c(1:n),y)
  ggplot(df) +
    geom_point(aes(x = df[,1], y = df[,2]),
               size = 0.01,
               color = "#FF3366") +
    labs(title = "Difference between the two Variances",
         x = "Index",
         y = "Difference")+
    theme_minimal()
}
Y1(x_vec)

var_YC <- function(v_x){
  ## v_x is a numerical vector of length greater than 2
  ## this function calculates the sample variance
  ## using the Youngs and Cramer algorithm
  T   <- v_x[1]
  RSS <- 0
  n   <- length(v_x)
  for (j in 2:n){
    T   <- T + v_x[j]
    RSS <- RSS + ((j * v_x[j] - T)^2) / (j*(j-1))
  }
  RSS /(n-1)
}

Y2 <- function(x){
  n <- length(x)
  y <- numeric(length = n)
  for(i in 2:n) {
    X    <- x_vec[1:i]
    y[i] <- var_YC(X) - var(X)
  }

  #plot(y, cex = 0.5)
  df <- data.frame(c(1:n),y)
  options(digits = 3)
  ggplot(df) +
    geom_point(aes(x = df[,1], y = df[,2]),
               size = 0.01,
               color = "#FF3366") +
    labs(title = "Difference between the two Variances",
         x = "Index",
         y = "Difference")+
    theme_minimal()
}
Y2(x_vec)


## Question 4: Linear Algebra
options(digits = 5)
tecator <- readxl::read_xls("tecator.xls")

X <- as.matrix(tecator[,-c(1,103)])
y <- as.matrix(tecator[,103])

A <- t(X) %*% X
b <- t(X) %*% y

solve.default(A,b)

X_scaled <- scale(X)
y_scaled <- scale(y)

A_new <- t(X_scaled) %*% X_scaled
b_new <- t(X_scaled) %*% y_scaled


kappa(A_new)
```